Madagascar Security Report

Implementacija i analiza Madagascar sistema za centralizovanu autorizaciju

Tina Mihajlović SV3/2020
Nemanja Vujadinovic SV28/2020
Nada Zarić SV63/2020

jun 2024.
Novi Sad, Srbija

Sadržaj

Uvod

Potrebno je implementirati mini verziju *Google*-ovog Zanzibar sistema za centralizovanu autorizaciju.

Implementirani mini Zanzibar je nazvan Madagascar. Pored samog API-a implementiran je klijent za demonstraciju autorizacije, MadagascarHub, kao web aplikacija za čuvanje i deljenje fajlova sa tri nivoa pristupa (viewer, editor, owner).

Pre implementacija sistema, identifikovani su bezbedonosni zahtevi i razvijen je model pretnji za isti. Nakon implementacije, izvršena je statička analiza koda sistema.

Bezbedonosni zahtevi

U nastavku su definisani bezbedonosni zahtevi Madagascar sistema, definisani na osnovu OWASP Application Security Verification Standard preporuka. U svakom poglavlju, ispod zahteva je opisana i implementacija onih zahteva koji su uspešno implementirani i integrisani u sistem. Za zahteve koji nisu ispunjeni u trenutnoj implementaciji, priložen je plan realizacije kao osnov za buduća proširenja sistema.

Autentifikacija - Madagascar

U našem sistemu ćemo koristiti API ključeve. API ključ će se dobijati uz pomoć endpointa GET http://localhost:4000/api-key. Ovaj endpoint će generisati API ključ i skladištiti ga u leveDB bazu podataka. Prije čuvanja u bazu, vrši se heširanje API ključa. Nemamo kompletno implementiran proces autentifikacije, ali ćemo u narednom tekstu opisati najbolju praksu za ovu funkcionalnost.

U svrhu implementacija autentifikacije uz pomoć API ključa za naš Madagascar, moguće je napraviti front klijent koji će obavljati registraciju samih korisnika i generisanje ključeva za korisnike. Možemo slijediti sledeće korake:

- korisnik se registruje na naš Madagascar sistem. Klijent je na primjer naša SpringBoot aplikacija koja želi koristiti usluge Madagascar sistema,
- nakon što je nalog kreiran klijent može generisati svoj API ključ. Prije čuvanja genrisanog API ključa u nekoj bazi, obavezno je da se izvrši transformacija ključa:
    - heširanje - ako želimo da naš klijent vidi samo jednom ključ (u toku generisanje) onda ćemo koristi heširanje. Kada jednom hešujemo ključ više nećemo biti u mogućnosti da dobavimo originalnu vrijednost ključa. Zbog toga je potrebno obavjestiti klijenta da sačuva generisani ključ na sigurnom mjestu, jer ako ga ne sačuva neće više biti u mogućnosti da korist taj ključ.
    - enkriptovanje - u slučaju da želimo omogućiti korisniku da može ponovo vidjeti svoj ključ, onda je potrebno da ga enkriptujemo,
- sigurno dostavljanje ključa - potrebno je koristiti HTTPS za prenos API ključa
- ako je potrebno, može se ograničiti uloga ključa. Ako korisnik može uraditi samo neke od funkcionalnosti sisitema uz pomoć nekog ključa, onda mu ne treba dati “svemogući”

ključ, nego samo ključ koji dozvoljava pristup dozvoljenim funkcionalnostima. Ovo se može odraditi dodavanje scope-ova u kojima će se definisati specifične permisije
- ograničiti vremenski rok trajanja ključeva i omogućiti rotaciju ključeva. Dakle ključevi bi se trebali perodično mijenjati i korisniku se treba omogućiti mehanizam generisanja novih ključeva i zamjene starih
- ograničiti broj zahtjeva sa određenim API ključevima kako bi se spriječili zlonamjerni napadi ili problemi u performansama

Autorizacija - Madagascar

Autorizacija u Madagascaru će se obavljati uz pomoć API ključeva. Prije obavljanja funkcija endpointa, provjeravaće se da li se u zaglavlju zahtjeva nalazi API ključ. Ključ se ne treba stavljati u URL da ne bi bio vidljiv. Ako ključa nema, ne dozvoljava se prstup funkcionalnostima. Ako ipak ključ postoji, provjerava se da li je dobiveni ključ validan. Ovo će se provjeravati tako što se dobiveni ključ hešuje i traži u levelDB. Ako se ne nalazi u bazi podataka, zahtjev se odbija, ako postoji u bazi podataka zahtjev se prihvata i obrađuje.

Moguća poboljšanja:

- dodati vrijeme trajanja API ključa
- ograničiti broj zahtjeva koje token može poslati
- koristiti HTTPS umjesto HTTP za komunikaciju

Sprječavanje DOS napada - Madagascar

DOS (Denial of Service) napadi su vrsta napada na računarske sisteme ili mreže koji imaju za cilj da onemoguće ili ometu pravilno funkcionisanje servisa ili resursa koje korisnici žele da koriste. U ovakvim napadima, napadači preplavljuju ciljani sistem velikim brojem zahteva ili podataka, preopterećujući kapacitete sistema i uzrokujući da stvarni korisnici ne mogu pristupiti resursima ili uslugama. Da bi se ovo spriječilo mogu se koristiti mehanizmi kao što su ograničenje broja zahtjeva, podizanje firewall-a ili uz pomoć IDS (Intrusion Detection System) sistema.

Ograničenje broja zahtjeva

U našem sistemu ćemo implementirati mehanizam za ograničenja broja zahtjeva. Ovo ćemo implementirati uz pomoć flask-limiter Python biblioteke. Uz pomoć ovog mehanizma možemo ograničiti broj zahtjeva koje će naša aplikacija (Madagascar) da primi u minuti, satu ili danu. Na ovaj način se spriječava preopterećenje aplikacije, čime se štite resursi i održava stabilnost aplikacije.

Firewall

Firewall nad aplikacijom je sigurnosni mehanizam koji štiti aplikaciju od neovlašćenog pristupa i napada sa spoljnih mreža. On funkcioniše kao barijera između unutrašnje mreže aplikacije i spoljnih mreža, filtrirajući ulazni i izlazni saobraćaj na osnovu definisanih pravila i politika bezbednosti. Firewall može blokirati ili dozvoliti specifične vrste saobraćaja, kao što su HTTP

zahtevi, TCP/UDP portovi ili IP adrese. Osim zaštite, firewalli često omogućavaju i monitoring saobraćaja kako bi detektovali sumnjive aktivnosti.

IDS (Intrusion Detection System)

IDS (Intrusion Detection System) je sigurnosni sistem koji nadgleda mrežni saobraćaj ili aktivnosti na sistemu kako bi otkrio neobične ili sumnjive obrasce koji mogu ukazivati na prisustvo napadača ili ranjivosti. IDS koristi različite tehnike kao što su heuristička analiza i analiza anomalija da bi identifikovao potencijalne pretnje.

Autentifikacija - SpringBoot

Prijava i registracija

Za autentifikaciju će se koristi korsničko ime i lozinka. Prilikom čuvanja lozinki u bazi podataka, koristiće se BCrypt64Encoder (OWASP 2.10.3, 2.10.4). Ovaj algoritam je dizajniran kao sporiji algoritam, kako bi bio otporniji na *brute-force* napade. Salt heša je sastavni dio algoritma (OWASP 2.4.1, 2.4.2). Prilikom registrovanja korisnik će moći vidjeti i zamaskirati lozinku (OWASP 2.1.12). Minimalna dužina lozinke će biti 8 karaktera, gdje mora postojati barem jedno velik slovo, jedno malo slovo i jedna cifra (OWASP 2.1.4, 2.1.8).

Moguća poboljšanja:

- omogućiti mehanizam obnove lozinke (OWASP 2.1.5)
- pri registrovanju, korisnik je bi trebao da potvrdi identitet unošenjem OTP šifre koja je poslata na email adresu, odnosno broj telefona (OWASP 2.5.6)
- šifra bi trebala imati barem 12 karaktera, a togođe se treba postaviti i gornja granica (OWASP 2.1.1, 2.1.2)
- ne bi trebalo da postoji ograničenje u pogledu karakera (OWASP 2.1.9)
- prije kreiranja lične lozinke, preporučeno je korišćenje inicijalne šifre date od sistema (OWASP 2.3.1)
- moguće je pored salta dodati peper vrijednost pr generisanju lozinke
- moguće je uvesti rotaciju lozinki

MFA (Multi-Factor Authentication)

Za poboljšanje autentifikacije bilo bi korisno koristiti i mehanizam kao što je MFA (OWASP 2.2.4). Ova metoda autentifikacije zahtijeva od korisnika da pruži više od jednog načina dokazivanja svog identiteta pri prijavi ili pristupu sistemu. Tradicionalno, to može uključivati kombinaciju nečega što korisnik zna (npr. lozinka), nečega što posjeduje (npr. mobilni uređaj s generisanim kodom) ili npr. biometrijski podatak kao što je otisak prsta. MFA povećava sigurnost jer iako jedan faktor može biti kompromitovan, drugi faktori još uvijek pružaju zaštitu.

ReCAPTCHA

Za poboljšanje autentifikacije bilo bi korisno koristiti i tehnologiju kao što je ReCAPTCHA (OWASP 2.2.1). Ovo je tehnologija koja se koristi da bi se razlikovali ljudi od automatizovanih sistema (botova). Obično se koristi u situacijama gdje se želi spriječiti automatsko slanje formi ili

zaštititi web resurse od zloupotrebe. ReCAPTCHA često zahtijeva od korisnika da riješi jednostavni test koji bi trebao biti lako rješiv za ljude, ali težak za računarske programe.

Autorizacija - SpringBoot

Kontrola pristupa korišćena u projektu bila je zasnovana na šemi Expression-based access control u Spring framework-u. Sistem ima dvije uloge: neautentifikovani korisnik i autentifikovani korisnik. Neautentifikovani korisnik ima mogućnost registracije na sistem i prijave na sistem. Nakon što se neautentifikovani korisnik prijav na sistem, postaje autentifikovani korisnik i ima prava pristupa svim funkcijama u sistemu. U procesu prijave na sistem, korisniku se šalje njegov JWT token i korisnik je obavezan uz svaki zahtjev slati i JWT token.

Kada korisnik želi promijeniti ili izbrisati dokument, provjerava se da li je on vlasnik tog dokumenta. Na ovaj način se spriječava da korisnik manipuliše dokumentima koji nisu njegovi i za koje nema prava pristupa.

Bezbedna Komunikacija

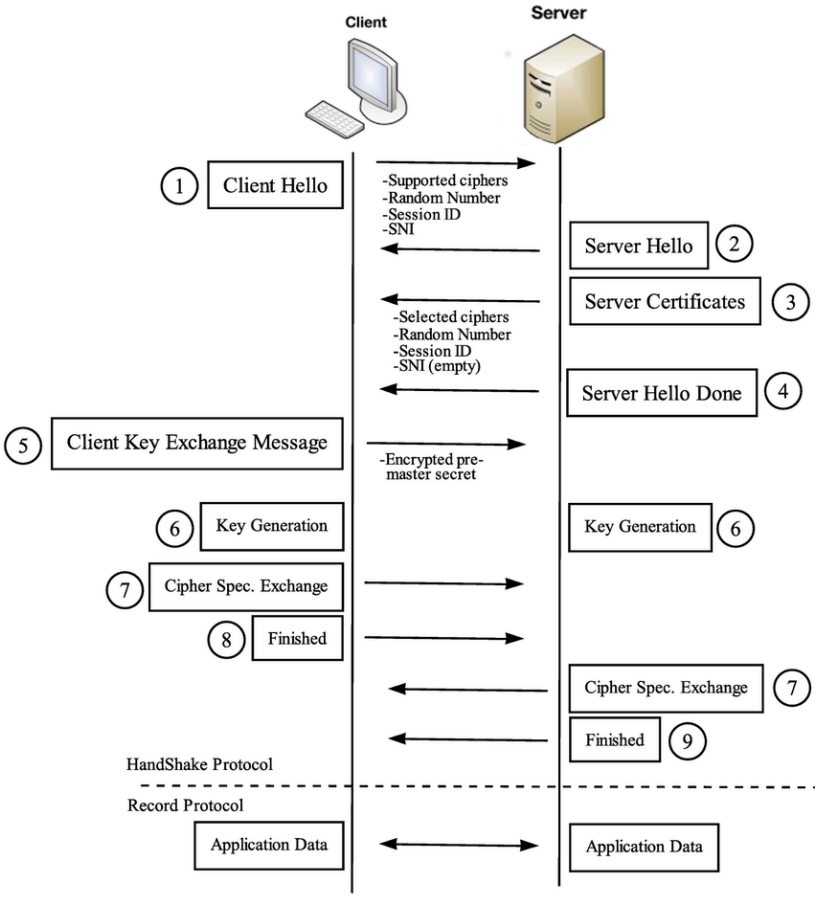
Naš sistem čini više zasebnih aplikacija koje međusobno komuniciraju:

- Madagascar (mini Zanzibar) - Flask aplikacija za centralizovanu autorizaciju, sa API-em isturenim za korišćenje od strane različitih klijenata.
- MadagascarHub - proof-of-concept klijent, aplikacija za čuvanje i deljenje fajlova. Razvijen za potrebe demonstracije funkcionalnosti Madagascar API-a. Sastoji se od:
    - SpringBoot backend - komunicira sa Madagascar API-em za CRUD nad namespace i ACL objektima, kao i upite o autorizaciji korisnika za pristup fajlovima.
    - Angular frontend - komunicira sa backend aplikacijom kao interfejs za krajnje korisnike.
- LevelDB - document-based key-value baza podataka u kojoj se skladište acl podaci koje koristi Madagascar.
- ConsulDB - key-value baza koja trči u svom Docker container-u i čuva informacije o namespace podacima koje koristi Madagascar.
- PostgreSQL - SQL baza podataka u kojoj SpringBoot backend čuva strukturirane podatke o korisnicima i fajlovima. Trči u zasebnom Docker container-u.

Prema OWASP specifikaciji, potrebno je koristiti TLS (Transport Layer Security) enkripciju za svu vrstu konekcija/saobraćaja od i ka sistemu (OWASP 9.1.1. i 9.2.2). TLS je kriptografski protokol koji obezbeđuje sigurnu komunikaciju preko mreže. TLS koristi asimetričnu kriptografiju za razmenu ključeva, simetričnu kriptografiju za šifrovanje podataka, i HMAC (Hash-based Message Authentication Code) za integritet poruka.

TLS radi tako što uspostavlja siguran kanal između klijenta i servera kroz proces poznat kao "handshake". Tokom ovog procesa, klijent i server razmenjuju informacije koje omogućavaju verifikaciju identiteta i dogovor o šifrovanju koje će koristiti za zaštitu komunikacije. Konkretno, TLS handshake uključuje sledeće ključne korake:

- Client Hello: Klijent šalje početnu poruku serveru sa listom podržanih TLS verzija, šifarskih skupova, i slučajnim brojem.
- Server Hello: Server odgovara sa izabranom TLS verzijom, šifarskim skupom, i svojim slučajnim brojem.
- Server Certificate: Server šalje svoj digitalni sertifikat klijentu, koji sadrži javni ključ servera.
- Server Key Exchange [opciono]: Server šalje dodatne informacije za razmenu ključeva.
- Client Key Exchange: Klijent generiše pre-master tajnu i šifruje je javnim ključem servera, zatim je šalje serveru.
- Change Cipher Spec: Obe strane šalju poruku koja označava da će buduća komunikacija biti šifrovana dogovorenim ključevima.
- Finished: Obe strane šalju poruku koja potvrđuje da je TLS handshake završen.



Dodatno, ukoliko je potrebna obostrana autentifikacija, server može da zahteva digitalni sertifikat klijenta. Pritom, od ključne je važnosti da digitalni sertifikati budu izdati od strane pouzdanog sertifikacionog tela (OWASP 9.2.1).

Postoje različite verzije TLS protokola, preporučuje se korišćenje najnovijih verzija - trenutno TLS 1.3. (OWASP 9.1.3).

Enkriptovanjem kanala komunikacije, branimo se od man-in-the-middle napada na sistem.

U našem sistemu, identifikovani su sledeći kanali komunikacije i načini obezbeđivanja istih:

- Madagascar API - SpringBoot backend
    - Svu komunikaciju vršiti preko HTTPS protokola (HTTP sa TLS enkripcijom)
        - Generisati sertifikat za Madagascar API
        - Konfigurisati Flask da koristi ssl/tsl
        - Generisati sertifikat za SprinBoot backend
        - Konfigurisati SpringBoot da koristi ssl/tls
    - Autentifikovati backend preko API key mehanizma (opisano ranije)
- Madagascar API - LevelDB
    - LevelDB je document-based baza podataka koja se nalazi u istom direktorijumu kao i Madagascar API, te nema mrežne komunikacije između ova dva servisa
    - Predlog za dodatno osiguravanje LevelDB baze bio bi enkritovanje fajlova u kojima se čuvaju torke.
        - Enkriptovanje fajlova u kojima se čuvaju podaci u LevelDB bazi dodatno osigurava podatke od neovlašćenog pristupa. Čak i ako neko dobije pristup fajlovima na disku, enkripcija osigurava da podaci ostanu nečitljivi bez odgovarajućeg ključa.
        - Ovaj pristup se može implementirati koristeći biblioteke za enkripciju u Pythonu, kao što je *cryptography*.
        - Prilikom upisa podataka u LevelDB, podaci se prvo šifruju, a zatim čuvaju, dok se pri čitanju podaci dešifruju, pa tek onda koriste.
        - Kako se LevelDB kod nas koristi za skladištenje ACL torki koje su ključ, dok vrednost jeste prazan niz bajtova, enkripcija bi se primenjivala na date torke tj. ključeve.
- Madagascar API - ConsulDB
    - Svu komunikaciju vršiti preko TLS enkripcije kako bi se osigurala bezbednost podataka u tranzitu.
    - Generisati i podesiti ConsulDB da koristi sertifikate.
- SpringBoot backend - PostgreSQL
    - Svu komunikaciju vršiti preko TLS enkripcije kako bi se osigurala bezbednost podataka u tranzitu.
    - Generisati sertifikate za PostgreSQL i konfigurisati ga da koristi TLS.
    - U SpringBoot backend-u konfigurisati konekciju ka PostgreSQL-u da koristi TLS.
- Angular frontend - SpringBoot backend
    - Komunikaciju između Angular frontend-a i SpringBoot backend-a vršiti preko HTTPS protokola.
    - Generisati i koristiti odgovarajuće sertifikate za osiguranje komunikacije.

Takođe, OWASP TLS Cheat Sheet preporučuje da se onemoguće svi protokoli/kanali za komunikaciju osim TLS.

Za potrebe projekta, generisan je self-signed sertifikat za Madagascar API pomoću *openssl* alata:

openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -sha256 -config C:\Users\Suncica\Documents\OpenSSL\openssl.cnf

U produkcionim sistemima, sertifikat bi bio generisan od strane pouzdanih CA, kako bi se izbegle začkoljice sa “Untrusted CA” upozorenjima kako na browser-ima tako i pri bilo kojoj vrsti komunikacije.

Zatim je Flask konfigurisan da koristi HTTP, kroz prosleđivanje *ssl_context* parametra pri pokretanju aplikacije:

```
cert_folder = os.getenv('CERT_FOLDER')

    ssl_context = (

        os.path.join(cert_folder, 'cert.pem'),

        os.path.join(cert_folder, 'key.pem')

    )

    app.run(host='127.0.0.1', port=4000, ssl_context=ssl_context)
```

Dodatno, primenjen je Flask-Talisman wrapper, kako bi se automatski sva HTTP komunikacija preusmerila na HTTPS:

```
Talisman(

    app,

    force_https=True

)
```

Na ovaj način je Madagascar API konfigurisan da prima samo HTTPS saobraćaj, što je potvrđeno kao funkcionalno iz Postman alata.

Međutim, prilikom komunikacije sa SpringBoot klijentom, HTTPS konekcija nije bila uspešna jer SpringBoot ne veruje self-signed sertifikatima out-of-the-box.

S toga je ekstrahovan digitalni sertifikat Flask aplikacije i manuelno ubačen u trust store Spring Boot aplikacije. Međutim, ponovo zbog specifičnosti klijenta, uvezivanje trust store-a povuklo je zahtev za kreiranjem key-store skladišta, te implementacije dolaznog HTTPs saobraćaja za SpringBoot. Kako ovo nije fokus projekta, odlučeno je da se SpringBoot kongifuracija

implementira kao buduće proširenje sistema. Posledično, zarad funkcionalnosti sistema kao celine, Madagascar API je vraćen na HTTP režim za potrebe demonstracije.

Komunikacija SpringBoot - Madagascar

Kao što je već navedeno, komunikacija se ostvaruje uz pomoć API ključeva i HTTP zahtjeva. Međutim, sigurniji način za komunikaciju bi bio HTTPS. HTTP podatke korsiti kao običan tekst (plant text) između klijenta i servera. To znači da se svi podaci, uključujući osjetljive informacije poput API ključeva, šalju nešifrovani i mogu biti presretnuti ili izmijenjeni od strane napadača. Međutim, HTTPS koristi SSL/TLS enkripciju za osiguranje komunikacije između klijenta i servera. HTTPS šifrira podatke koji se prenose, čime se štite od presretanja i manipulacije.

Logovanje

Logovanje je ključna komponenta za praćenje aktivnosti sistema, dijagnostiku problema i detekciju potencijalnih napada. Prema OWASP preporukama, logovanje u aplikacijama treba da bude dizajnirano i implementirano tako da obezbedi sveobuhvatne informacije o radu sistema, sa dva glavna cilja:

1. Debugging - reprodukcija toka događaja radi otklanjanja buggova u sistemu.
2. Security logging - dokumentovanje identifikatora korisnika i bitnih akcija radi:
    a. Korišćenje log zapisa za automatsku detekciju intruzija kroz specijalizovane sisteme.
    b. Forenzičku analizu i istragu nakon (pokušaja) napada.

Pritom, važno je obratiti pažnju na sledeće smernice:

- Izbegavanje logovanja previše ili premalo informacija:
    - Logovati relevatne informacije neophodne za kasniju analizu i rekonstrukciju događaja - vremensku oznaku, izvor IP adresu i korisnički ID (OWASP 7.1.4).
    - Izbegavati logovanje privatnih ili poverljivih podataka (lozinke, sesija ID-ovi, brojevi kreditnih kartica, JMBG) (OWASP 7.1.1 i 7.1.2)
- Logovanje za detekciju i odgovor na upade:
    - Logovati aktivnosti koje ukazuju na potencijalno zlonamerno ponašanje (unos podataka van očekivanog opsega, pokušaji modifikacije zaštićenih podataka) (OWASP 7.2.1 i 7.2.2.).
    - Obeležavanje sumnjivih aktivnosti kao visoko rizičnih i implementacija odgovora na identifikovane napade (invalidacija korisničkih sesija, zaključavanje korisničkih naloga).
- Siguran dizajn logovanja:
    - Sanitizacija i enkodiranje ulaznih podataka pre logovanja kako bi se neutralisali zlonamerni karakteri (OWASP 7.3.1).
    - Podešavanje permisija za log fajlove i audit promena kako bi se zaštitili od neovlašćenih promena (OWASP 7.3.3.)

U okviru OWASP Top 10 Proactive Controls, definiše se C9: Implement Security Logging and Monitoring koji skreće pažnju na potencijalne probleme koje logovi unose u sistem, među kojima je log injection.

Kako bismo zaštitili sistem od injektovanja nepostojećih logova, koda i komandi u naše logove, potrebno je implementirati sledeće mehanizme:

- Sanitizacija i enkodiranje ulaznih podataka pre logovanja kako bi se neutralisali zlonamerni karakteri (OWASP 7.3.1) - uklanjanje ili transformisanje opasnih karaktera.
- Sigurnosni formateri za logove koji neće omogućiti umetanje zlonamernih karaktera.

Kako je Madagascar API centralna komponenta sistema, predlaže se sledeći mehanizam logovanja:

- Primena sanitizacije i enkodiranja ulaza:
  - Pre logovanja, sanitizovati i enkodirati sve ulazne podatke kako bi se neutralisali zlonamerni karakteri.
- Detalji logovanja zahteva:
  - Logovati sve dolazne zahteve sa informacijama o IP adresi, korisničkom ID-u, metodi zahteva i URL-u.
- Logovanje ključnih operacija:
  - Logovati uspešne i neuspešne pokušaje dodavanja, provere i brisanja ACL unosa.
  - Beležiti visoko rizične aktivnosti, kao što su neuspešni pokušaji pristupa zaštićenim resursima.

Kako je u pitanju Flask aplikacija, predlaže se *logging* Python biblioteka.

*Logovi su bitan alaz za povećanje stepena bezbednosti čitavog sistema, te bi slične smernice valjalo primeniti i na ostale komponente, shodno njihovom tipu i nameni.

U aplikaciji je implementirano logovanje za Madagascar API, korišćenjem logging python biblioteke. Koriste se dva logger-a:

1. Application logger - za loggovanje infromacija o svakom pristiglom zahtevu ka API-u, kao i uspešnosti tih zahteva.
2. Security logger - za logovanje događaja koji potencijalno ugrožavaju bezbednost aplikacije (sve akcije koje izazivaju grešku).

Format logova je uniforman:

```
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s - IP: %(ip)s
```

Logovi se prosleđuju konzoli na prikaz i upisuju u fajl za trajno skladištenje.

Dodatno, message komponenta svih logova je enkriptovana korišćenjem *cryptography* Python biblioteke, kroz klasu Fernet. Fernet koristi AES u Cipher Block Chaining (CBC) modu sa

128-bitnim ključem za enkripciju podataka. Na ovaj način su podaci u logovima dostupni (dekriptovanjem) samo korisnicima sa ključem, čak iako maliciozni napadač uspe da dođe do log fajla ili ispisa konzole.

Ključ za enkriptovanje se čuva u .env fajlu projekta. Zarad dodatne bezbednosti, moguće je proširiti sistem tako da se ključ čuva u sigurnijim skladištima - poput hardverskih sigurnosnih modula (HSM) ili usluga za upravljanje ključevima (KMS) koje pružaju cloud provajderi.

Validacija

Validacija na klijentskoj strani

Za validaciju na klijentskoj strani aplikacije, korišćeni su ručno implementirani validatori, bazirani na *Regex* izrazima. Ovi validatori su umetnuti pri svakom polju za unos od strane korisnika. Primer jedne validatorske funkcije dat je ispod:

```
export function nameRegexValidator( control: AbstractControl): { [key: string]: boolean } | null {
   const regex = /^([a-zA-Zčćđžš ]*)$/;
   if (control.value !== undefined && !regex.test(control.value)) {
       return { nameRegexError: true };
   }
   return null;
 }
```

Takođe, svako polje je ulančano sa validatorom za obavezno popunjavanje polja.

Validacija na serverskoj strani

Po OWASP-u, validacija na klijentskoj strani nike dovoljna, te je potrebno implementirati i validaciju na serverskoj strani.

Za validaciju na serverskoj strani aplikacije, takođe su korišćeni *Regex* izrazi. Dodatno, korišćene su anotacije da bi se polja postavila kao obavezna. Kod DTO objekata (i klasa entiteta), dodate su anotacije za *Regex* proveru. Primer anotacija za jedno polje klase DTO dat je ispod:

```
@NotEmpty(message="is required")
@Pattern(regexp = "^([a-zA-Zčćđžš ]*)$", message="format is not valid")
```

Za mejl, korišćena je ugrađena anotacija `@Email` čime smo se osigurali od loše ručno konstruktovanih *Regex* izraza.

Korišćenje *Regex* izraza

Kao što je već pomenuto, *Regex* izrazi su korišćeni kako i na klijentskom sloju aplikacije, tako i na serverskom. Ovo je podržano kao dobra praksa od strane OWASP-a.

OWASP takođe priča o validaciji i sanetizaciji HTML-a, u slučaju kada aplikacija treba da prihvati HTML od korisnika, te da u tim slučajevima, *Regex* izrazi nisu najbolje rešenje. Takvih slučajeva kod nas nema. Mana naših *Regex* izraza biće objašnjenja u nastavku.

Neočekivan unos korisnika (Mass Assignment)

OWASP nalaže da se korisnički unos ne vezuje direktno (OWASP 5.1.2) i kao rešenje govore o upotrebi DTO objekata. Naš sistem podržava DTO objekte na mnogo nivoa, a naročito u slučajevima kada se prenose informacije koje korisnik unosi - tada je svaki zahtev propraćen DTO objektima.

Dodatno

Po OWASP-u, strukturirani podaci bi trebalo da budu validirani po šemi (5.1.4). Ovakav slučaj u našem sistemu nalazimo za podatke mejla, gde se izričito traži standardna mejl šema - example + @ + mail + . + domain.

Po OWASP-u, upiti za bazu podataka treba da sadrže parametrizovane upite (5.3.4). Naš sistem sadrži nekolicinu upita za bazu, ali su u svakom slučaju korišćeni parametrizovani upiti.

Revizija

Po OWASP-u, validacija unosa predstavlja tehniku bezbednosti kojim bi se mogli sprečiti određeni napadi, ali u isto vreme, ona nije primarni metod za prevenciju od *XSS, SQL injection* i drugih napada. S tim u vezi, potrebno bi bilo revidirati delove koda koji sadrže obradu unosa, te postaviti dodatne metode zaštite.

Takođe, po OWASP-u, loši *Regex* izrazi mogu dovesti do *ReDoS*-a. Primer takvog *Regex* izraza bio bi `([a-zA-Z]+)*` i kao takav, mogao bi biti korišćen pri polju za ime korisnika. Naš *Regex* izraz za ime ne bi bio problematičan po pitanju *ReDoS*-a, zbog neimanja znaka +. Ipak, OWASP savetuje da *Regex* izrazi često znaju da budu teško održivi ili nerazumljivi, te bi trebalo razmisliti o njihovom delimičnom uklanjaju i zameniti ih proverom koristeći izvorni kod.

Rukovanje greškama i *Exception*-i

Po OWASP-u, stvari koje se pogrešno rade pri rukovanju greškama tiču se problema curenja informacija. Greške bi trebalo da budu logovane, ali tako da daju dovoljno (i ne više) informacija nego što je potrebno *Support* timu, *QA* timu i ostalima. Takođe, zapis grešaka treba da bude uniforman, kratak i jasan. Rukovanje greškama trebalo bi da centralizovano.

Naše greške ne sadrže osetljive informacije, već su njihove poruke generičke - nepostojani korisnik, mejl ili šifra. Takođe, u sistemu ne postoji ispis *stack trace*-a *exception*-a, što OWASP nalaže.

Takođe, razlike u porukama grešaka pri istom zahtevu svedene su na minimum. Na primer, pri prijavi na sistem, ukoliko korisnik nije ispravno uneo mejl ili lozinku, greška sadrži sledeću poruku: Wrong username or password! Ovako, kada korisnik jednom unese pogrešnu šifru, a sledeći put mejl, dobiće se ista poruka o grešci čime dajemo manje informacija potencijalnom napadaču.

Greške u našem sistemu su jednostavne - Something is (not) valid!.

Centralizovanje pri rukovanju greškama implementiralo je tako da jedna metoda, tj. funkcija, pri kojoj može doći do *exception*-a, njega i baci, a ostale metode ne moraju da brinu o tome. Na primer, sledeća metoda baca exception u slučaju nepostojećeg korisnika:

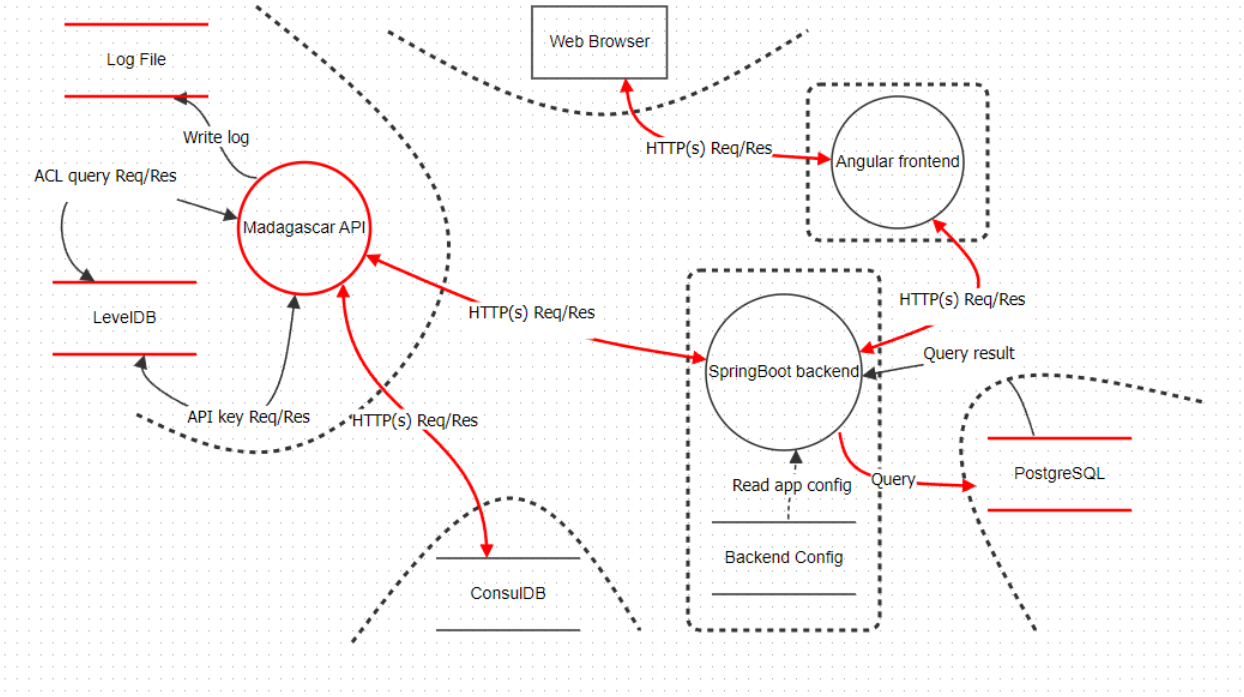
```
@Override

public User getUserByEmail(String email) {

User user = allUsers.findByEmail(email).orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, "User does not exist!"));

  return user;

}
```

Ovu metodu koriste i druge, ali ne brinu o bacanju greške, do koje ukoliko dođe, sav dalji rad će biti obustavljen. Takođe, implementiran je i *RestAuthenticationEntryPoint* čime je omogućeno lakše korišćenje *exception*-a.

Model pretnji

Modelovanje pretnji *(Threat modeling)* je sistematski pristup identifikaciji i evaluaciji potencijalnih pretnji i ranjivosti u informacionim sistemima ili aplikacijama. Ključni koraci uključuju identifikaciju vrednih resursa i osetljivih podataka, mapiranje potencijalnih pretnji koje bi mogle ugroziti te resurse, procenu ranjivosti i njihovih uticaja te predlaganje odgovarajućih mera zaštite. Za modelovanje pretnji korišćen je Threat Dragon alat.

Korišćen je STRIDE pristup modelovanju pretnji. Skraćenica STRIDE predstavlja šest osnovnih kategorija pretnji: "Spoofing" (falsifikacija identiteta), "Tampering" (manipulacija podacima), "Repudiation" (poricanje), "Information disclosure" (otkrivanje informacija), "Denial of Service" (nedostupnost usluge) i "Elevation of Privilege" (povećanje privilegija).



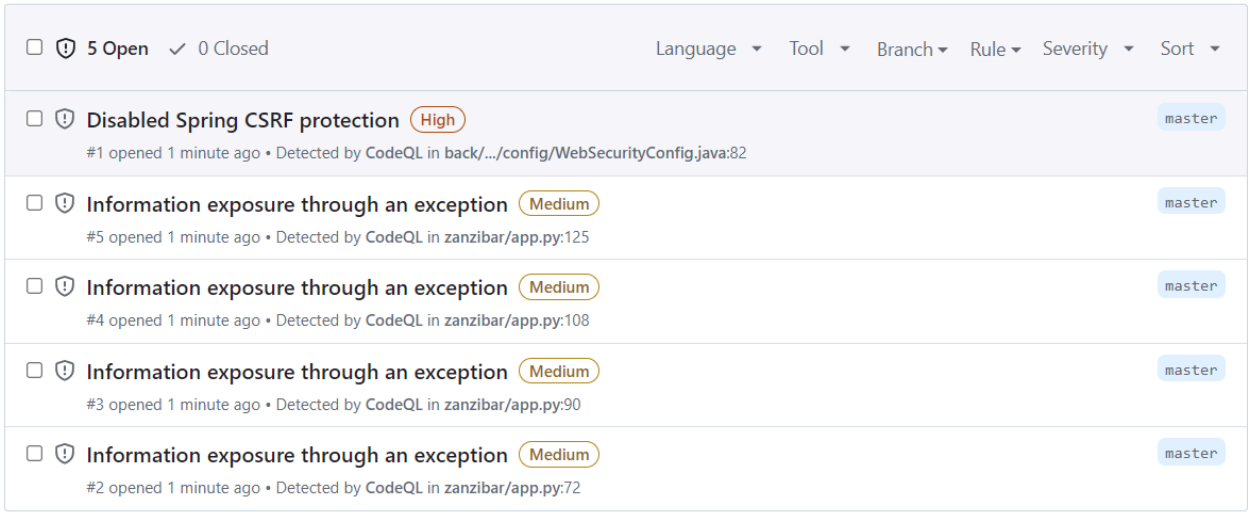
*Detaljan opis pretnji moguće je pronaći na repozitorijumu projekta, na lokaciji: *ThreatDragonModels/Threat Model*

Statička analiza koda

Statička analiza koda je proces provere izvornog koda softvera bez njegovog izvršavanja. Ova tehnika identifikuje potencijalne greške, ranjivosti i druge probleme u kodu pre nego što se aplikacija pokrene ili pušta u produkciju. Statička analiza se obično vrši pomoću automatizovanih alata koji skeniraju kod radi pronalaženja nedostataka u pravilima programiranja, potencijalnih sigurnosnih slabosti ili performansi. Ova metoda omogućava razvojnim timovima da unapred detektuju i isprave probleme kako bi poboljšali kvalitet i pouzdanost softvera.

Za statičku analizu Madagascar stistema, korišćen je *Github CodeQL* alat.

Analizom kompletne baze koda, CodeQL je prijavio sledećih 5 ranjivosti sistema:



Prva ranjivost, *Disabled Spring CSRF protection*, nije relevatna za sistem, budući da je onemogućavanje ove protekcije standardna procedura kada se za SpringBoot koristi JWT token za autentifikaciju korisnika.

Preostale četiri ranjivosti zapravo govore o jednoj te istoj ranjivosti, na 4 različita mesta: ostavljeno je da se pri grešci vrati celokupna poruka o grešci korisniku, bez filtriranja i kontrole podataka koji se na taj način daju na uvid:

Ranjivost je rešena uvođenjem nove klase izuzetaka *MadagascarException*, koja prosleđuje poruke o grešci koje generiše naša logika unutar Madascar API-a (dakle kontrolisane poruke u kojima provereno nema osetljivih podataka).

Sve ostale, neproverene greške, se hvataju uniformno kroz *except Exception* blok, pri čemu se prosleđuje uvek ista poruka o grešci “Internal server error”, a detalji prikazuju kroz logove.